

# Style Guide

[L CS50 Manual \(/\)](#)

## Table of Contents

[Comments](#)

[Conditions](#)

[Switches](#)

[Functions](#)

[Indentation](#)

[Loops](#)

[for](#)

[while](#)

[do ... while](#)

[Pointers](#)

[Variables](#)

There's no one, right way to stylize code. But there are definitely a lot of wrong (or, at least, bad ways). Even so, CS50 does ask that you adhere to the conventions below so that we can reliably analyze your code's style. Similarly do companies typically adopt their own, company-wide conventions for style.

## Comments

Comments make code more readable, not only for others (e.g., your TF) but also for you, especially when hours, days, weeks, months, or years pass between writing and reading your own code. Commenting too little is bad. Commenting too much is bad. Where's the sweet spot? Commenting every few lines of code (i.e., interesting blocks) is a decent rule of thumb. Try to write comments that address one or both of these questions:

1. What does this block do?
2. Why did I implement this block in this way?

Within functions, use "inline comments" and keep them short (e.g., one line), else it becomes difficult to distinguish comments from code, even with [syntax highlighting](http://en.wikipedia.org/wiki/Syntax_highlighting) ([http://en.wikipedia.org/wiki/Syntax\\_highlighting](http://en.wikipedia.org/wiki/Syntax_highlighting)). No need to write in full sentences, but do leave one space between the `//` and your comment's first character, as in:

```
// convert Fahrenheit to Celsius
float c = 5.0 / 9.0 * (f - 32.0);
```

In other words, don't do this:

```
//convert Fahrenheit to Celsius
float c = 5.0 / 9.0 * (f - 32.0);
```

Or this:

```
// Convert Fahrenheit to Celsius.
float c = 5.0 / 9.0 * (f - 32.0);
```

Atop your .c and .h files should be multi-line comments that summarize what your program (or that particular file) does along with, perhaps, your name and that of the file, as in:

```
/**
 * hello.c
 *
 * David J. Malan
 * malan@harvard.edu
 *
 * Says hello to the world.
 */
```

Notice how:

1. the first line starts with `/**`;
2. the last line ends with `*/`; and
3. all of the asterisks (`*`) between those lines line up perfectly in a column.

Atop each of your functions (except, perhaps, `main`), meanwhile, should be multi-line comments that summarize what your function, as in:

```
/**
 * Returns  $n^2$  ( $n$  squared).
 */
int square(int n)
{
    return n * n;
}
```

---

## Conditions

Conditions should be styled as follows:

```
if (x > 0)
{
    printf("x is positive\n");
}
else if (x < 0)
{
    printf("x is negative\n");
}
else
{
    printf("x is zero\n");
}
```

Notice how:

1. the curly braces line up nicely, each on its own line, making perfectly clear what's inside the branch;
2. there's a single space after each `if`;
3. each call to `printf` is indented with 4 spaces;
4. there are single spaces around the `>` and around the `<`; and
5. there isn't any space immediately after each `(` or immediately before each `)`.

To save space, some programmers like to keep the first curly brace on the same line as the condition itself, but we don't recommend, as it's harder to read, so don't do this:

```
if (x < 0) {
    printf("x is negative\n");
} else if (x < 0) {
    printf("x is negative\n");
}
```

And definitely don't do this:

```
if (x < 0)
{
    printf("x is negative\n");
}
else
{
    printf("x is negative\n");
}
```

---

## Switches

Declare a `switch` as follows:

```
switch (n)
{
    case -1:
        printf("n is -1\n");
        break;

    case 1:
        printf("n is 1\n");
        break;

    default:
        printf("n is neither -1 nor 1\n");
        break;
}
```

Notice how:

1. each curly brace is on its own line;
2. there's a single space after `switch`;

3. there isn't any space immediately after each `(` or immediately before each `)`;
4. the switch's cases are indented with 4 spaces;
5. the cases' bodies are indented further with 4 spaces; and
6. each `case` (including `default`) ends with a `break`.

---

## Functions

Be sure to define `main`, in accordance with [C99](http://en.wikipedia.org/wiki/C99) (<http://en.wikipedia.org/wiki/C99>), with:

```
int main(void)
{

}
```

Or with:

```
int main(int argc, char* argv[])
{

}
```

However, if using the [CS50 Library \(/library/\)](#), it's fine to define `main` with

```
int main(int argc, string argv[])
{

}
```

since `string` is just a `typedef` (i.e., synonym) for `char*`.

Do not declare `main` with:

```
int main(int argc, char** argv)
{

}
```

or with:

```
int main()
{

}
```

or with:

```
void main()
{

}
```

or with:

```
main()
{

}
```

As for your own functions, be sure to define them similarly, with each curly brace on its own line and with the return type on the same line as the function's name, just as we've done with `main`.

---

## Indentation

Indent your code four spaces at a time to make clear which blocks of code are inside of others. If you use your keyboard's Tab key to do so, be sure that your text editor's configured to convert tabs (`\t`) to four spaces, else your code may not print or display properly on someone else's computer, since `\t` renders differently in different editors. (If using the [CS50 Appliance \(/appliance/\)](#), it's fine to use Tab for indentation, rather than hitting your keyboard's space bar repeatedly, since we've preconfigured `gedit` and other programs to convert `\t` to four spaces.) Here's some nicely indented code:

```
// print command-line arguments one per line
printf("\n");
for (int i = 0; i < argc; i++)
{
    for (int j = 0, n = strlen(argv[i]); j < n; j++)
    {
        printf("%c\n", argv[i][j]);
    }
    printf("\n");
}
```

---

## Loops

### for

Whenever you need temporary variables for iteration, use `i`, then `j`, then `k`, unless more specific names would make your code more readable:

```
for (int i = 0; i < LIMIT; i++)
{
    for (int j = 0; j < LIMIT; j++)
    {
        for (int k = 0; k < LIMIT; k++)
        {
            // do something
        }
    }
}
```

If you need more than three variables for iteration, it might be time to rethink your design!

### while

Declare `while` loops as follows:

```
while (condition)
{
    // do something
}
```

Notice how:

1. each curly brace is on its own line;
2. there's a single space after `while`;
3. there isn't any space immediately after the `(` or immediately before the `)`; and
4. the loop's body (a comment in this case) is indented with 4 spaces.

## do ... while

Declare `do ... while` loops as follows:

```
do
{
    // do something
}
while (condition);
```

Notice how:

1. each curly brace is on its own line;
2. there's a single space after `while`;
3. there isn't any space immediately after the `(` or immediately before the `)`; and
4. the loop's body (a comment in this case) is indented with 4 spaces.

---

## Pointers

When declaring a pointer, write the `*` next to the type, as in:

```
int* p;
```

Don't write it next to the variable's name, as in:

```
int *p;
```

This convention can lead to ambiguity in some contexts, but we think, overall, it's clearer when first learning pointers.



---

# Variables

Because CS50 uses [C99](http://en.wikipedia.org/wiki/C99) (<http://en.wikipedia.org/wiki/C99>), do not define all of your variables at the very top of your functions but, rather, when and where you actually need them. Moreover, scope your variables as tightly as possible. For instance, if `i` is only needed for the sake of a loop, declare `i` within the loop itself:

```
for (int i = 0; i < LIMIT; i++)
{
    printf("%i\n", i);
}
```

Though it's fine to use variables like `i`, `j`, and `k` for iteration, most of your variables should be more specifically named. If you're summing some values, for instance, call your variable `sum`. If your variable's name warrants two words (e.g., `is_ready`), put an underscore between them, a convention popular in C though less so in other languages.

If declaring multiple variables of the same type at once, it's fine to declare them together, as in:

```
int quarters, dimes, nickels, pennies;
```

Just don't initialize some but not others, as in:

```
int quarters, dimes = 0, nickels = 0 , pennies;
```

Also take care to declare pointers separately from non-pointers, as in:

```
int* p;
int n;
```

Don't declare pointers on the same line as non-pointers, lest it be ambiguous as to whether the latter was meant to be the former, as in:

```
int* p, n;
```